IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of :

Maiko TARUKI et al. :

Serial No. NEW : **Attn: APPLICATION BRANCH**

Filed September 23, 2003 : Attorney Docket No. 2003_1310A

SIMULATOR FOR SOFTWARE DEVELOPMENT AND RECORDING MEDIUM HAVING SIMULATION PROGRAM RECORDED THEREIN

THE COMMISSIONER IS AUTHORIZED TO CHARGE ANY DEFICIENCY IN THE FEES FOR THIS PAPER TO DEPOSIT ACCOUNT NO. 23-0975

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-280827, filed September 26, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Maiko TARUKI et al.

By _____
David M. Ovedovitz
Registration No. 45,336
Attorney for Applicants

DMO/jmj
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
September 23, 2003

日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００２年　９月２６日

出願番号
Application Number: 特願２００２－２８０８２７

[ST.10/C]: [ＪＰ２００２－２８０８２７]

出願人
Applicant(s): 松下電器産業株式会社

２００３年　４月２５日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号　出証特２００３－３０３０３６３

【書類名】　特許願

【整理番号】　2838240102

【提出日】　平成14年 9月26日

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/45

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　樽木　麻衣子

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　中村　剛

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　近藤　孝宏

【特許出願人】

【識別番号】　000005821

【氏名又は名称】　松下電器産業株式会社

【代理人】

【識別番号】　100097179

【弁理士】

【氏名又は名称】　平野　一幸

【手数料の表示】

【予納台帳番号】　058698

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　0013529

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　シミュレータ、そのシミュレータをコンピュータ読み取り可能に記録した記録媒体

【特許請求の範囲】

【請求項１】高級言語で記述されたソースコードをコンパイルするコンパイラと、

前記ホストプロセッサとは異なる、ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きと、

この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きと

を含む

ライブラリとを備え、

前記ソースコードは、このライブラリを使用して記述される、シミュレータ。

【請求項２】前記ハードウエア構成要素には、ターゲットプロセッサの演算器と、ターゲットプロセッサのメモリコントローラと、ターゲットプロセッサのレジスタとが含まれる、請求項１記載のシミュレータ。

【請求項３】前記ライブラリは、

ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きを定義するハードウエアモデルライブラリと、

前記ハードウエアモデルライブラリにおける、関数又は手続きを利用して、ターゲットプロセッサの命令に対応する関数又は手続きを、高級言語で定義する命令セットライブラリと

を備える、請求項１から２記載のシミュレータ。

【請求項４】前記ターゲットプロセッサの命令には、ＡＤＤ命令、ＳＵＢ命令、ＡＮＤ命令、ＯＲ命令、ＬＤ命令、ＳＴ命令、ＳＥＴ命令及びＭＯＶ命令が含まれる、請求項１から３記載のシミュレータ。

【請求項５】前記ライブラリには、ターゲットプロセッサにおける実行サイクル数、消費電力のうち、一方又は双方を計算する関数又は手続きが含まれている、請求項１から４記載のシミュレータ。

【請求項６】ターゲットプロセッサにおける、実行サイクル数、消費電力のうち、一方又は双方は、変更可能となっている、請求項１から４記載のシミュレータ。

【請求項７】前記ライブラリには、ターゲットプロセッサにおけるコードサイズを計算する関数又は手続きが含まれている、請求項１から６記載のシミュレータ。

【請求項８】オブジェクトコードをフェッチする命令フェッチ手段と、

フェッチされたオブジェクトコードをデコードする命令デコード手段と、

デコードされた命令をホストプロセッサ上で実行する実行手段と、

高級言語で記述されたソースコードを読み込んでオブジェクトコードを生成する翻訳装置と、

前記ホストプロセッサとは異なる、ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きと、

この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きと

を含むライブラリとを備え、

前記ソースコードは、このライブラリを使用して記述される、シミュレータ。

【請求項９】請求項１から８記載のシミュレータが、コンピュータ読み取り可能に記録された記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、ターゲットプロセッサ向け組み込みソフトウエアの開発に適した、シミュレータ及びその関連技術に関するものである。

【０００２】

【従来の技術】

ターゲットプロセッサ向け組み込みソフトウエアを開発する際、一般に、ターゲットプロセッサとは互換性のない、ホストプロセッサ上で動作するソフトウエア開発環境が用いられる。

【０００３】

ここで、本明細書において、「ホストプロセッサ」とは、このソフトウエア開発環境を動作させ、開発と、開発の結果物であるソフトウエアの検証とに、使用するプロセッサをいう。

【０００４】

また、「ターゲットプロセッサ」とは、ホストプロセッサとは異なるプロセッサであり、この開発の結果物であるソフトウエアを実行するプロセッサをいう。

【０００５】

そして、ホストプロセッサとターゲットプロセッサとは、ソフトウエア互換性がない。また、このような、結果物としてのソフトウエアは、専らターゲットプロセッサ上でのみ正常に動作し、ホストプロセッサ上では正常に動作しない。

【０００６】

また、本明細書にいう「シミュレータ」が生成するソフトウエアは、上記結果物であるソフトウエアを模したソフトウエアであるが、ターゲットプロセッサ上で動作するものではなく、ホストプロセッサ上で動作するものである。

【０００７】

このように、ホストプロセッサのソフトウエア開発環境を用いて、ホストプロセッサとは異なるターゲットプロセッサ向けのソフトウエアを開発する場合、次に述べるような問題点がある。

【０００８】

ここで近年、このようなソフトウエアを開発する場合であっても、ソフトウエアが大規模になるに伴い、アセンブラ言語主体による開発から、Ｃ／Ｃ＋＋等の高級言語による開発へ、シフトしている。

【０００９】

これは、高級言語を用いたソフトウエア開発を行えば、データの保持、転送、演算等の処理を、ターゲットプロセッサのアセンブラレベルの命令やレジスタ、メモリ等のリソースに依存しない形式で記述でき、可読性、汎用性、開発効率に優れているためである。

【００１０】

ここで特に、組み込み向けソフトウエア等では、システムのパフォーマンスを最大限引き出すため、プロセッサの能力を極限まで高める、最適化が求められる。

【００１１】

しかしながら、高級言語によるソフトウエア開発では、コンパイラ性能の問題から高級言語からアセンブラコードに変換する際に、冗長なコードが生成され、ソフトウエアのコードサイズや実行速度に影響を及ぼす場合がある。

【００１２】

したがって現在でも、信号処理などの処理負荷が大きい部分に関しては、高級言語による開発に加え、ターゲットプロセッサのアセンブラ言語による開発が行われている。ところが、これでは、ソフトウエア開発の効率が落ちてしまう。

【特許文献１】

特開２０００－３３０８２１号公報（第４－８頁）

【００１３】

【発明が解決しようとする課題】

組み込み向けソフトウェア開発では、ソフトウェア開発の大規模化・複雑化に伴い、その開発工数が増大する傾向にある。加えて、検証工数、シミュレーション時間の増大も著しい。

【００１４】

そして、ターゲットプロセッサの完成前に、ターゲットプロセッサを入手できないまま、ソフトウェア開発に着手しなければならないケースが多い。

【００１５】

また、ソフトウェア開発環境が十分に整備されていない段階でも、プログラムのパフォーマンス解析を行いたいものである。しかしながら、ＬＳＩの大規模化・複雑化に、シミュレータの速度が対応できておらず、短期間で高精度な情報を得ることは難しい。

【００１６】

さらに、従来技術では、コンパイラやシミュレータなどのソフトウェア開発環境の整備にかかる工数が多く、この開発環境を用いた、目的のソフトウェア開発

の着手が遅れがちとなる。

【００１７】

そこで本発明は、ソフトウエアを効率良く開発できる、コンパイラ型及びインタープリタ型の２種類の、シミュレータを提供することを目的とする。より詳しくは、本発明は、アセンブラレベルの最適化（コードサイズ、実行速度の両面で）も含め、全て高級言語で開発された１つのソフトウェアを、これら２種類のシミュレータで、共用して、シミュレーションできる、シミュレータを提供するものである。

【００１８】

【課題を解決するための手段】

請求項１記載のシミュレータは、高級言語で記述されたソースコードをコンパイルするコンパイラと、ホストプロセッサとは異なる、ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きと、この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きとを含むライブラリとを備え、ソースコードは、このライブラリを使用して記述される。

【００１９】

この構成において、ターゲットプロセッサのハードウエア構成要素を、ライブラリにおいて関数又は手続きにより、モデル化し、ソースコードをこのライブラリを利用して記述するようにしている。これにより、コンパイラ型のシミュレータを構成できる。

【００２０】

コンパイラ型のシミュレータが出力するオブジェクトコードは、命令フェッチや命令デコードなどのプロセスを経ずに実行できるため、実行速度が高速である。

【００２１】

また、ソースコードが全て高級言語で記述されるため、移植性や可読性に優れた、開発を行えるし、アセンブラレベルの最適化も含め、全て高級言語でのソフトウェア開発が可能となる。

【００２２】

しかも、後述するように、このソースコードは、インタプリタ型のシミュレータにも共用でき、コンパイラ型、インタプリタ型の２種類のシミュレータを開発するにあたり、部品を共通化でき、開発効率が向上する。

【００２３】

請求項２記載のシミュレータでは、ハードウエア構成要素には、ターゲットプロセッサの演算器と、ターゲットプロセッサのメモリコントローラと、ターゲットプロセッサのレジスタとが含まれる。

【００２４】

この構成により、ターゲットプロセッサの主要な構成要素である、演算器、メモリコントローラの振る舞いを、モデル化により、表現できる。

【００２５】

請求項３記載のシミュレータでは、ライブラリは、ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きを定義するハードウエアモデルライブラリと、ハードウエアモデルライブラリにおける、関数又は手続きを利用して、ターゲットプロセッサの命令に対応する関数又は手続きを、高級言語で定義する命令セットライブラリとを備える。

【００２６】

この構成において、ライブラリを、ハードウエアモデルライブラリと、命令セットライブラリとに分けているため、ライブラリの取り扱いが容易になる。即ち、ハードウエアモデルライブラリを、後述するインタプリタ型のシミュレータと共有すると共に、コンパイラ型のシミュレータには、命令セットライブラリが追加されることになる。

【００２７】

言い換えれば、この構成に係るコンパイラ型のシミュレータは、ハードウエアモデルライブラリに、命令セットライブラリを追加するだけで実現可能となり、翻訳装置などが不要であるため、インタプリタ型のシミュレータに比べ、簡易に構成できる。このため、シミュレータの開発工数が減少し、早期提供が可能となる。

【００２８】

請求項４記載のシミュレータでは、ターゲットプロセッサの命令には、ＡＤＤ命令、ＳＵＢ命令、ＡＮＤ命令、ＯＲ命令、ＬＤ命令、ＳＴ命令、ＳＥＴ命令及びＭＯＶ命令が含まれる。

【００２９】

この構成により、ターゲットプロセッサの主要な命令をカバーできる。

【００３０】

請求項５のシミュレータでは、ライブラリには、ターゲットプロセッサにおける実行サイクル数、消費電力のうち、一方又は双方を計算する関数又は手続きが含まれている。

【００３１】

この構成により、ライブラリに含まれる、サイクル数、消費電力等を計算する関数又は手続きによる、計算結果を出力させれば、シミュレーション終了時にアセンブラプログラムの実行サイクル数、消費電力等を得ることができ、コンパイラ型のシミュレータにおいて、パフォーマンス解析が可能になる。

【００３２】

請求項６記載のシミュレータでは、ターゲットプロセッサにおける、実行サイクル数、消費電力のうち、一方又は双方は、変更可能となっている。

【００３３】

この構成により、実行サイクル数、消費電力等を意図的に変更することにより、様々な条件下での、シミュレーションを行える。

【００３４】

請求項７記載のシミュレータでは、ライブラリには、ターゲットプロセッサにおけるコードサイズを計算する関数又は手続きが含まれている。

【００３５】

この構成により、コードサイズによる最適化や、パフォーマンス解析を行える。

【００３６】

請求項８記載のシミュレータは、オブジェクトコードをフェッチする命令フェ

ッチ手段と、フェッチされたオブジェクトコードをデコードする命令デコード手段と、デコードされた命令をホストプロセッサ上で実行する実行手段と、高級言語で記述されたソースコードを読み込んでオブジェクトコードを生成する翻訳装置と、ホストプロセッサとは異なる、ターゲットプロセッサのハードウエア構成要素を、高級言語でモデル化する関数又は手続きと、この関数又は手続きを利用して、ターゲットプロセッサの命令に対応して高級言語で定義された関数又は手続きとを含むライブラリとを備え、ソースコードは、このライブラリを使用して記述される。

【００３７】

この構成により、インタプリタ型のシミュレータを構成できる。しかも、このインタプリタ型のシミュレータのソースコードは、上述したコンパイラ型のシミュレータのそれと、共用でき、コンパイラ型、インタプリタ型、都合２種類のシミュレータ上で動作させるプログラムを一本化でき、プログラムの開発効率を向上できる。

【００３８】

【発明の実施の形態】

以下、図面を参照しながら、本発明の実施の形態を説明する。なお、具体的な説明に先立ち、本明細書にいう高級言語及びその関数又は手続きについて、簡単に述べる。

【００３９】

以下の各形態では、高級言語として、Ｃ言語を用いる。Ｃ言語の仕様上、戻り値のない（ｖｏｉｄ型）の処理を、”手続き”とは呼ばず、”関数”と呼ぶ習慣がある。しかしながら、他の高級言語、例えばＰａｓｃａｌでは、戻り値のない手続き（ｐｒｏｃｅｄｕｒｅ）と、戻り値を持つ関数（ｆｕｎｃｔｉｏｎ）とを、厳格に区別している。

【００４０】

ここで、本明細書にいう高級言語は、Ｃ言語に限定されず、その上位のＣ＋＋言語、Ｐａｓｃａｌなども含む。

【００４１】

したがって、本明細書にいうライブラリでは、ターゲットプロセッサのハードウエア構成要素は、”関数又は手続きにより”モデル化されており、この関数又は手続きを利用し、ターゲットプロセッサの命令に対応する、”関数又は手続き”は、高級言語で定義されているということになる。

【００４２】

（実施の形態１）

次に、本形態の具体的構成を説明する。さて、図１は、本発明の実施の形態１におけるソフトウエア開発環境のブロック図である。大きく分けて、この開発環境は、コンパイラ型のシミュレータ１０２と、インタプリタ型のシミュレータ１０８との、２種類のシミュレータから構成される。

【００４３】

勿論、本発明のシミュレータは、コンパイラ型あるいはインタプリタ型のシミュレータを、単独に構成することもできるし、このように、単独に構成したシミュレータも、本発明に包含される。

【００４４】

図１に示しているように、これら、コンパイラ型のシミュレータ１０２、インタプリタ型のシミュレータ１０８は、いずれも同一のハードウエアモデルライブラリ１０１を利用する。

【００４５】

ハードウエアモデルライブラリ１０１は、後にソースコード例を示して詳述するように、ターゲットプロセッサのハードウエア構成要素（演算器、メモリコントローラ、レジスタなど）を、Ｃ言語でモデル化した、関数、変数等を定義する。

【００４６】

また、命令セットライブラリ１０５は、ハードウエアモデルライブラリ１０１における、関数等を利用して、ターゲットプロセッサの命令に対応する関数等を、Ｃ言語で定義する。

【００４７】

本形態のライブラリは、ハードウエアモデルライブラリ１０１と、命令セット

ライブラリ１０５とからなる。

【００４８】

また、このライブラリには、コンパイラ型のシミュレータ１０２によるソフトウェアのパフォーマンス解析を可能にするため、サイクル数や消費電力、リソースの使用状況などを計算する関数が設けられている。

【００４９】

このうち、ハードウエアモデルライブラリ１０１は、コンパイラ型のシミュレータ１０２、インタプリタ型のシミュレータ１０８の、２種類のソフトウェアシミュレータに共通に適用される。

【００５０】

すなわち、ターゲットプロセッサ内のハードウェア機能を実現する関数群を提供するライブラリを用意し、そのライブラリ関数、変数を、コンパイラ型、インタプリタ型のシミュレータの構成要素として利用する。

【００５１】

次に、コンパイラ型のシミュレータ１０２の概要について説明する。ソースコード１０３は、ハードウエアモデルライブラリ１０１、命令セットライブラリ１０５の定義を利用して記述された、Ｃ言語のプログラムである。このソースコード１０３は、インタプリタ型のシミュレータ１０８のソースコードでもある。

【００５２】

コンパイラ１０４は、ホストプロセッサ上で動作する、Ｃ言語のコンパイラである。このコンパイラ１０４は、通常のＣコンパイラであれば十分である。そして、コンパイラ１０４は、ソースコード１０３をコンパイルして、オブジェクトファイル１０６を出力する。

【００５３】

命令セットライブラリ１０５は、後述するこのライブラリのソースコードを既にコンパイルしてあり、オブジェクトファイルの形態になっている。

【００５４】

リンカ１００は、オブジェクトファイル１０６と、命令セットライブラリ１０５（オブジェクトファイル）とを、リンクして、機械語のオブジェクトコード１

０７を出力する。このオブジェクトコード１０７は、ホストプロセッサ上で動作するものであり、ターゲットプロセッサ上では、一般には、動作しない。

【００５５】

次に、インタプリタ型のシミュレータ１０８の概要について、説明する。まず、翻訳装置１１２は、Ｃ言語で記述したプログラムを、ターゲットプロセッサ用のオブジェクトコード１１３に翻訳する。

【００５６】

オブジェクトコード１１３は、実行時に、命令フェッチ手段１０９により、１命令ずつ読み出され、命令デコード手段１１０によって解読され、実行手段１１１により実行される。

【００５７】

次に、図２を用いて、本形態で想定するターゲットプロセッサの構成を説明する。図２は、本発明の実施の形態１で使用するターゲットプロセッサのブロック図である。勿論、図示しているターゲットプロセッサは例示にすぎず、他の形態を採るターゲットプロセッサについても、本発明は、同様に適用できる。

【００５８】

図２に示すターゲットプロセッサは、次の要素を有する。まず、命令レジスタ（ＩＲ）２０１は、実行中の命令を保持するレジスタである。

【００５９】

プログラムカウンタ（ＰＣ）２０５は、現在実行中の命令のアドレスを保持するレジスタである。

【００６０】

レジスタ群２０６は、後述するメモリから読み出したデータや演算結果を保持するものであり、本形態では、４個の１６ビットのレジスタで構成される。これらのレジスタを、それぞれ、レジスタＲ０（２０９）、レジスタＲ１（２１０）、レジスタＲ２（２１１）、レジスタＲ３（２１３）と呼ぶ。

【００６１】

図４（ａ）に示すように、演算器（ＡＬＵ）２０７は、加減算（ＡＤＤ、ＳＵＢ）、論理演算（ＡＮＤ、ＯＲ）計４種類の１６ビットバイナリ演算を実行する

ユニットであり、２つのデータ入力（ＩＮ１、ＩＮ２）、１つのデータ出力（ＯＵＴ）、ならびに１つの制御信号（ＣＴＬ）を持つ。

【００６２】

また、図４（ｂ）に示すように、メモリコントローラ２０８は、データメモリＡ（２０３）、データメモリＢ（２０４）へのアクセスを制御するユニットであり、アドレス値によりアクセスするデータメモリを選択する。

【００６３】

なお、図４（ｂ）において、ＡＤ信号は、アクセスするメモリへのアドレス値を表す。ＤＢ信号は、ポインタである。メモリコントローラ２０８は、リード時に、メモリ変数から読み出したデータを、ＤＢ信号が指すポインタへデータを格納し、ライト時に、ＤＢ信号が指すポインタ内のデータを読み出しメモリ変数へ格納する。ＲＷ信号は、リード／ライトを表す制御信号である。

【００６４】

また、図１において、以上のターゲットプロセッサには、バス２００を介して、次のメモリが接続される。

【００６５】

命令メモリ（ＩＭＥＭ）２０２は、プログラムを格納するためのメモリであり、データメモリＡ（２０３）、データメモリＢ（２０４）は、ターゲットプロセッサが、演算に使用するデータを格納するためのメモリである。

【００６６】

次に、図３を用いて、ターゲットプロセッサの命令セットを説明する。図３（ａ）は、本発明の実施の形態１におけるターゲットプロセッサの命令セットの説明図である。

【００６７】

図３（ａ）に示すように、このターゲットプロセッサは、ＡＤＤ命令、ＳＵＢ命令、ＡＮＤ命令、ＯＲ命令、ＬＤ命令、ＳＴ命令、ＳＥＴ命令、ＭＯＶ命令の、８種類の命令セットを持つ。

【００６８】

また、ターゲットプロセッサは、レジスタの指定について、図３（ｂ）に示す

規則に従う。

【００６９】

ＡＤＤ命令では、ターゲットプロセッサは、指定された２つのオペランドレジスタ内に格納されているデータに対して加算を行い、結果を第１オペランドレジスタへ格納する。

【００７０】

ＳＵＢ命令では、ターゲットプロセッサは、指定された２つのオペランドレジスタ内に格納されているデータに対して減算を行い、結果を第１オペランドレジスタへ格納する。

【００７１】

ＡＮＤ命令では、ターゲットプロセッサは、指定された２つのオペランドレジスタ内に格納されているデータに対し、論理積を計算し、結果を第１オペランドレジスタへ格納する。

【００７２】

ＯＲ命令では、ターゲットプロセッサは、指定された２つのオペランドレジスタ内に格納されているデータに対し、論理和を計算し、結果を第１オペランドレジスタへ格納する。

【００７３】

ＬＤ命令では、ターゲットプロセッサは、第２オペランドレジスタで指定したレジスタに格納されているデータを、データメモリＡ（２０３）またはデータメモリＢ（２０４）のアドレスとして、データメモリ内のデータを第１オペランドレジスタへ格納する。

【００７４】

ＳＴ命令では、ターゲットプロセッサは、第２オペランドレジスタに格納されているデータを、データメモリＡ（２０３）またはデータメモリＢ（２０４）のアドレスに、第１オペランドレジスタ内のデータを格納する。

【００７５】

ＳＥＴ命令では、ターゲットプロセッサは、第２オペランドで指定した即値を第１オペランドレジスタへ格納する。

【００７６】

ＭＯＶ命令では、ターゲットプロセッサは、第２オペランドレジスタに格納されているデータを第１オペランドレジスタへ格納する。

【００７７】

次に、図５、図６を用いて、図２に示すターゲットプロセッサの各要素をモデル化した、ハードウエアモデルライブラリ１０１の構成例について述べる。

【００７８】

ここで、図５は、本発明の実施の形態１におけるハードウエアモデルライブラリのヘッダファイルの例示図、図６は、本発明の実施の形態１におけるハードウエアモデルライブラリの実装ファイルの例示図である。

【００７９】

なお、以下図２、図３において名付けた要素に対応する、プログラムの要素には、同じ名前を付与する。例えば、図２の命令メモリ（ＩＭＥＭ）２０２に対応する、プログラムの要素には、ＩＭＥＭという名前を付与する。

【００８０】

さて、ここでは、図５のようなヘッダファイルで、変数や関数の宣言を行い、その内容は、図６の実装ファイルで記述している。しかしながら、ヘッダファイルと実装ファイルとに分離しなくとも良く、１つのファイルで全て記述してしまってもかまわない。

【００８１】

図４及び図５に示すように、配列ＩＭＥＭ、ＤＭＥＭＡ、ＤＭＥＭＢは、それぞれ、命令メモリ２０２、データメモリＡ（２０３）、データメモリＢ（２０４）を表し、それぞれのサイズ分の要素を持つ。勿論、メモリをモデル化するデータ構造は、配列でなくとも良く、リストなど、他の周知の構造を使用して差し支えない。

【００８２】

変数ＩＲ、ＰＣ、Ｒ０、Ｒ１、Ｒ２、Ｒ３は、それぞれ、命令レジスタ（ＩＲ）２０１、プログラムカウンタ（ＰＣ）２０５、レジスタＲ０（２０９）、レジスタＲ１（２１０）、レジスタＲ２（２１１）、レジスタＲ３（２１３）を表し

、１６ビットの整数型であるｓｈｏｒｔ型の変数で宣言する。

【００８３】

演算器（ＡＬＵ）２０７は、図４（ａ）に示すように、変数ＩＮ１、ＩＮ２、ＯＵＴ、ＣＴＬの４つの引数を持つ、ＡＬＵ関数で表現する。

【００８４】

ここで、変数ＩＮ１、ＩＮ２は２つのデータ入力を表し、ｓｈｏｒｔ型とする。変数ＯＵＴは、データ出力を表し、ｓｈｏｒｔ型のポインタとする。変数ＣＴＬは、制御信号を表し、ｉｎｔ型とする。なお、ＡＬＵ関数４１０は、ハードウェアと同じ演算精度となるようＣ言語で記述してある。

【００８５】

メモリコントローラ（ＭＥＭＣ）２０８は、図４（ｂ）に示すように、変数ＡＤ、ＤＢ、ＲＷの３つの引数を持つＭＥＭＣ関数で、表現する。

【００８６】

ここで、変数ＡＤは、アクセスするメモリへのアドレス値を表す。変数ＤＢは、ポインタであり、ＭＥＭＣ関数は、リード時に、メモリ変数から読み出したデータを、変数ＤＢの指すポインタへデータを格納し、ライト時に、変数ＤＢの指すポインタ内のデータを読み出しメモリ変数へ格納する。なお、変数ＲＷは、リード／ライトを表す制御信号である。

【００８７】

本形態では、以上の９個の変数（配列も含めている）ＩＭＥＭ、ＤＭＥＭＡ、ＤＭＥＭＢ、ＩＲ、ＰＣ、Ｒ０、Ｒ１、Ｒ２、Ｒ３と、２つのライブラリ関数（ＡＬＵ関数、ＭＥＭＣ関数）を含めて、ハードウエアモデルライブラリ１０１を構成している。

【００８８】

次に、図７を用いて、命令セットライブラリ１０５の実装例について、説明する。但し、オブジェクトファイルにする前の、ソースコードレベルで述べる。

【００８９】

図３（ａ）を参照しながら上述したように、ターゲットプロセッサは、ＡＤＤ命令、ＳＵＢ命令、ＡＮＤ命令、ＯＲ命令、ＬＤ命令、ＳＴ命令、ＳＥＴ命令、

ＭＯＶ命令の、８種類の命令セットを持っている。そこで、本形態の命令セットライブラリ１０５では、これらの命令に、一対一に対応する次の関数を用意している。

【００９０】

図７に示しているように、ＡＤＤ関数、ＳＵＢ関数、ＡＮＤ関数、ＯＲ関数は、それぞれ、２つの引数ＲＳ１、ＲＳ２を持つ関数として定義され、ハードウエアモデルライブラリ１０１のライブラリ関数であるＡＬＵ関数４１０を、呼び出すことにより、演算が実施されるようになっている。

【００９１】

ここで、引数ＲＳ１、ＲＳ２は、それぞれ命令セットの第１、第２オペランドレジスタに対応する。

【００９２】

また、ＬＤ関数、ＳＴ関数は、２つの引数ＲＳ１、ＲＳ２を持つ関数として定義され、ハードウエアモデルライブラリ１０１のライブラリ関数であるＭＥＭＣ関数４１５を、呼び出すことにより、メモリのコントロールが実施されるようになっている。

【００９３】

ここで、引数ＲＳ１、ＲＳ２は、それぞれ命令セットの第１、第２オペランドレジスタに対応する。

【００９４】

ＳＥＴ関数は、２つの引数ＲＤ、ＩＭＤを持つ関数として定義され、引数ＩＭＤの値を、＊ＲＳ１に代入する。

【００９５】

ＭＯＶ関数は、２つの引数ＲＤ、ＲＳを持つ関数として定義され、引数ＲＳ２の値を、＊ＲＳ１に代入する。

【００９６】

本形態の命令セットライブラリ１０５には、以上の８つの命令に対応した関数が用意されている。そして、図７の実装ファイルを、プログラム用のソースコードでインクルードしやすいように、図８（ａ）に示すような、ヘッダファイルを

用意しておく。

【００９７】

次に、図８を用いて、図２に示すソースコード１０３の例について述べる。まず、図８のソースコード１０３では、冒頭で、図８（ａ）に示したヘッダファイルをインクルードしており、以下、図７で実装した、命令セットライブラリ１０５の各関数を利用できる。

【００９８】

そして、ｍａｉｎ関数内において、ＳＥＴ関数、ＭＯＶ関数等を呼び出して、所望の処理を行うようにしている。なお、このソースコード１０３には、命令セットライブラリ１０５に含まれる関数のみしか記述できないものではなく、通常のＣ言語で利用できる関数等を記述することができる。

【００９９】

また、このようにすると、ソースコード１０３→命令セットライブラリ１０５→ハードウエアモデルライブラリ１０１の順に、呼び出しが行われることになる。

【０１００】

このソースコード１０３を、パーソナルコンピュータやワークスステーションなどのホストプロセッサ上で、コンパイラ１０４を用いて、コンパイルし実行することにより、ターゲットプロセッサの動作をシミュレーションすることができる。

【０１０１】

次に、図９を用いて、図２に示す、インタプリタ型のシミュレータ１０８について説明を追加する。まず、翻訳装置１１２について、説明する。

【０１０２】

図９（ａ）には、本形態のインタプリタ型のシミュレータ１０８による処理の流れを示してあり、図９（ｂ）には、それに対応して、アセンブラ９０２で処理を行う場合の流れを示している。

【０１０３】

さて、図９（ａ）に示すように、本形態の翻訳装置１１２は、ソースコード１

０３（コンパイラ型のシミュレータ１０２と同じもの）を読み込んで、オブジェクトコード１１３を出力する必要がある。

【０１０４】

一方、ソースコード１０３（Ｃ言語）に対応する、アセンブラプログラム９０１がある時、アセンブラ９０２が、このアセンブラプログラム９０１をアセンブルして、オブジェクトコード９０３を出力する。

【０１０５】

ここで、これらのオブジェクトコード１１３、９０３が一致すれば、目的を達成できるわけである。

【０１０６】

即ち、この翻訳装置１１２を簡単に構成するならば、翻訳装置１１２は、

（処理１）ソースコード１０３を入力し、これを、アセンブラプログラム９０１に置換する、

（処理２）翻訳装置１１２は、アセンブラ９０２相当の機能を内蔵し、置換したアセンブラプログラム９０１を、アセンブルする、

という、２つの処理を行うようにすればよい。

【０１０７】

ここで、（処理１）は、＊、？をワイルドカードとして、

（規則１）「＃ｉｎｃｌｕｄｅ　＊」という行を削除する、

（規則２）「ｍａｉｎ（）＊｛」を「ｍａｉｎ：」に置換する、

（規則３）「ＳＥＴ（＆＊，？）；」を「ＳＥＴ　＊，？」に置換する、

．．．（規則ｎ）という、置換規則に従えば、単なる文字列の置換により、対応できる。また、（処理２）もアセンブラを流用すればよいから、簡単に対応できる。

【０１０８】

但し、翻訳装置１１２は、（処理１）と（処理２）とを分離して実施する必要はなく、一度に処理を完了するようにしても良い。

【０１０９】

次に、図１０を用いて、図２の命令フェッチ手段１０９、命令でコード手段１

１０、実行手段１１１について説明する。

【０１１０】

図１０は、本発明の実施の形態１におけるインタプリタ型のシミュレータの実装ファイルの例示図である。

【０１１１】

まず、冒頭で、ハードウエアモデルライブラリ１０１のヘッダファイルをインクルードしている。そして、状態を示す変数ｓｔａｔｅの型を、「Ｆｅｔｃｈ」、「Ｄｅｃｏｄｅ」、「Ｅｘｅｃ」の３つだけ、列挙して宣言している。もちろん、変数ｓｔａｔｅが「Ｆｅｔｃｈ」の場合は、命令フェッチ手段１０９が動作すべきことを示し、同様に、変数ｓｔａｔｅが「Ｄｅｃｏｄｅ」、「Ｅｘｅｃ」の場合は、それぞれ、命令デコード手段１１０、実行手段１１１が動作すべきことを示す。また、サイクル数を記憶する変数ｃｙｃｌｅを、ｉｎｔ型で宣言し導入している。

【０１１２】

そして、ｍａｉｎ関数において、はじめに変数ｃｙｃｌｅを「０」とし、変数ｓｔａｔｅを「Ｆｅｔｃｈ」にしてから処理を開始する。

【０１１３】

次のｗｈｉｌｅ分において、ｅｘｅｃ関数を呼び出し、最後に変数ｃｙｃｌｅの値を、標準出力している。これにより、実行サイクル数を計測できる。

【０１１４】

ｅｘｅｃ関数では、呼び出される毎に、変数ｃｙｃｌｅが「＋１」され、以下、ｓｗｉｔｃｈ文において、変数ｓｔａｔｅによって、分岐して処理が実行される。

【０１１５】

変数ｓｔａｔｅが「Ｆｅｔｃｈ」の場合は、命令フェッチ手段１０９に相当する処理（ｃａｓｅ　Ｆｅｔｃｈ：以下ｃａｓｅ　Ｄｅｃｏｄｅ：の直前まで）が実行される。

【０１１６】

変数ｓｔａｔｅが「Ｄｅｃｏｄｅ」の場合は、命令デコード手段１１０に相当

する処理（ｃａｓｅ　Ｄｅｃｏｄｅ：以下ｃａｓｅ　Ｅｘｅｃ：の直前まで）が実行される。

【０１１７】

変数ｓｔａｔｅが「Ｅｘｅｃ」の場合は、実行手段１１１に相当する処理（ｃａｓｅ　Ｄｅｃｏｄｅ：以下）が実行される。

【０１１８】

ここで、以上の各処理において、配列ＩＭＥＭやＡＬＵ関数など、ハードウエアモデルライブラリ１０１で定義された、変数、配列、関数などが利用されている点に注目されたい。

【０１１９】

＜実施の形態１の効果＞

以上のように、ターゲットプロセッサの構成要素を単位として、機能を関数などでモデル化したライブラリを要し、これを利用することにより、コンパイラ型１０２、インタプリタ型のシミュレータ１０８の開発効率が向上する。

【０１２０】

命令セットライブラリ１０５は、単なるソースコードを用意し、それをコンパイルしておくだけで良いから、インタプリタ型のシミュレータ１０８に比べ構成が容易であり、開発工数が減少し、インタプリタ型１０８に比べ早期提供が可能となる。

【０１２１】

また、命令セットライブラリ１０５の関数を用いて記述した、ソースコード１０３から、アセンブラコードで記述したコードをアセンブルして得られるオブジェクトコードと同じオブジェクトコードを生成する翻訳装置１１２を組み入れることで、２種類のシミュレータ上で動作させるプログラムを一本化できる。

【０１２２】

（実施の形態２）

以下、実施の形態１との相違点について説明する。

まず、図１１に示すように、ハードウエアモデルライブラリ１０１において、サイクル数ならびに消費電力を表す変数ｃｙｃｌｅ、ｐｏｗｅｒを追加する。

【０１２３】

また、ＡＬＵ関数において、実際のハードウエアで規定されているＡＬＵ演算にかかるサイクル数と消費電力を、それぞれ変数ｃｙｃｌｅ、ｐｏｗｅｒに加える処理を追加している。

【０１２４】

ＭＥＭＣ関数において、実際のハードウエアで規定されているメモリアクセスに要するサイクル数ならびに消費電力をそれぞれ、変数ｃｙｃｌｅ、ｐｏｗｅｒに加える処理を追加している。

【０１２５】

次に、図１２に示すように、命令セットライブラリ１０５において、ＳＥＴ関数、ＭＯＶ関数につき、命令実行に要するサイクル数と消費電力を計算するために、変数ｃｙｃｌｅ、ｐｏｗｅｒを加算する処理を追加している。

【０１２６】

さらに、図１３に示すように、ソースコード１０３において、変数ｃｙｃｌｅ、ｐｏｗｅｒの値を、ｐｒｉｎｔｆ関数を用いて標準出力する行を追加している。勿論、標準出力せず、変数ｃｙｃｌｅ、ｐｏｗｅｒの値を、ファイルに書き出すなど種々変更して差し支えない。

【０１２７】

これにより、プログラムの実行に要したサイクル数と消費電力に関する情報を得ることができる。

【０１２８】

図示していないが、実施の形態２における、インタプリタ型のシミュレータ１０８では、以上と同様に、図１０のｍａｉｎ関数において、変数ｐｏｗｅｒの値をｐｒｉｎｔｆ関数を用いて標準出力する機能等を追加すると良い。

【０１２９】

これにより、インタプリタ型のシミュレータ１０８においても、プログラムの実行に要した消費電力に関する情報を得ることができる。

【０１３０】

＜実施の形態２の効果＞

実施の形態１の効果に加え、ライブラリにサイクル数ならびに消費電力を計算する処理を組み込み、対応する変数を出力させることで、シミュレーション終了時にアセンブラプログラムの実行サイクル数ならびに消費電力を得ることができ、コンパイラ型、インタプリタ型の、２種類のシミュレータにおいてパフォーマンス解析が可能になる。

【０１３１】

（実施の形態３）

実施の形態３においては、実施の形態２との相違点のみを説明する。

まず、図１４に示すように、命令セットライブラリ１０５において、サイクル数を表す変数ｃｙｃｌｅ、消費電力を表す変数ｐｏｗｅｒ、コードサイズを表す変数ｃｏｄｅを、追加する。

【０１３２】

また、本形態では、図１５（ａ）に表でまとめているように、ＡＤＤ命令、ＳＵＢ命令など、命令毎にインデックスをふり、変数ｃｙｃｌｅの増分、変数ｐｏｗｅｒの増分を定め、これらの増分を、配列ｃｙｃｌｅ＿ｔｂｌ［］、配列ｐｏｗｅｒ＿ｔｂｌ［］に格納する。勿論、配列でない他の構造でこれらの増分を格納しても良い。

【０１３３】

そして、図１４に示すように、命令セットライブラリ１０５において、ＡＤＤ関数、ＳＵＢ関数、ＬＤ関数、ＳＴ関数、ＭＯＶ関数において、これらの変数ｃｙｃｌｅ、ｐｏｗｅｒ、ｃｏｄｅを加算する処理を追加する。ここで、変数ｃｙｃｌｅ、ｐｏｗｅｒを計算する際は、各命令実行時に要するデータが格納されている配列ｃｙｃｌｅ＿ｔｂｌ［］、ｐｏｗｅｒ＿ｔｂｌ［］の値を利用する。

【０１３４】

なお、図１５（ｂ）のソースコード例に示すように、配列ｃｙｃｌｅ＿ｔｂｌ［］、ｐｏｗｅｒ＿ｔｂｌ［］の値のデータを、ファイル（図示の例では”ｔａｂｌｅ”）に格納しておき、初期化時に、ファイルから配列ｃｙｃｌｅ＿ｔｂｌ［］、ｐｏｗｅｒ＿ｔｂｌ［］にロードするようにすると良い。

【０１３５】

なお、このｉｎｉｔ関数は、シミュレーション開始前に予め実行する初期化関数である。

【０１３６】

＜実施の形態３の効果＞

実施の形態２の効果に加え、命令実行に要するサイクル数、消費電力に関する情報を、外部（本形態ではファイル）から与える仕組みを採用することにより、これらの情報を容易に変更することが可能となり、様々なケースに応じたシミュレーションが可能となる。

【０１３７】

以上の説明では、オブジェクト指向を利用しないライブラリの記述例を説明したが、例えば、Ｃ＋＋言語等により、ハードウエアモデルライブラリ１０１、命令セットライブラリ１０５を、よりスマートに記述しても良い。

【０１３８】

また、実施の形態１から３に記載の、コンパイラ型のシミュレータ１０２、インタプリタ型のシミュレータ１０８等は、「プログラムをコンピュータ読み取り可能に記憶した記録媒体」（ＣＤ－ＲＯＭ、ＦＤ、ハードディスク等）に格納して提供できるし、パーソナルコンピュータ、ワークスステーションなどにインストールしてから、この装置ごと供給することもできる。

【０１３９】

ここで、本明細書にいう「プログラムをコンピュータ読み取り可能に記憶した記録媒体」には、複数の記録媒体にプログラムを分散して配布する場合を含む。また、このプログラムが、オペレーティングシステムの一部であるか否かを問わず、種々のプロセスないしスレッド（ＤＬＬ、ＯＣＸ、ＡｃｔｉｖｅＸ等（マイクロソフト社の商標を含む））に機能の一部を肩代わりさせている場合には、肩代わりさせた機能に係る部分が、記録媒体に格納されていない場合も含む。

【０１４０】

図１には、スタンドアロン形式のシステムを例示したが、サーバー／クライアント形式にしても良い。つまり、１つの端末機のみに、本明細書に出現する全ての要素が含まれている場合の他、１つの端末機がクライアントであり、これが接

続可能なサーバないしネットワーク上に、全部又は一部の要素が実存していても、差し支えない。

【０１４１】

さらには、図１のほとんどの要素をサーバー側で持ち、クライアント側では、例えば、WWWブラウザだけにしても良い。この場合、各種の情報は、通常サーバ上にあり、基本的にネットワークを経由してクライアントに配布されるものだが、必要な情報が、サーバ上にある時は、そのサーバの記憶装置が、ここにいう「記録媒体」となり、クライアント上にある時は、そのクライアントの記録装置が、ここにいう「記録媒体」となる。

【０１４２】

さらに、この「プログラム」には、コンパイルされて機械語になったアプリケーションの他、上述のプロセスないしスレッドにより解釈される中間コードとして実存する場合や、少なくともリソースとソースコードとが「記録媒体」上に格納され、これらから機械語のアプリケーションを生成できるコンパイラ及びリンカが「記録媒体」にある場合や、少なくともリソースとソースコードとが「記録媒体」上に格納され、これらから中間コードのアプリケーションを生成できるインタープリタが「記録媒体」にある場合なども含む。

【０１４３】

【発明の効果】

本発明によれば、ターゲットプロセッサの構成要素単位で機能を、関数等によって、モデル化しておくことにより、コンパイラ型、インタプリタ型の２種類のシミュレータを開発するにあたり、部品を共通化でき、開発効率が向上する。

【０１４４】

また、アセンブラレベルの最適化も含め、全て高級言語でのソフトウェア開発が可能となる。

【０１４５】

さらに、翻訳装置を追加することで、コンパイラ型、インタプリタ型の、２種類のシミュレータに用いるソースコードを、一本化できる。

【０１４６】

また、ライブラリにサイクル数、消費電力ならびにコードサイズを計算する処理を組み込むことにより、コンパイラ型のシミュレータにおいてもパフォーマンス解析が可能になる。

【０１４７】

また、これらの情報を容易に変更し、様々なケースに応じたシミュレーションが可能となる。

【図面の簡単な説明】

【図１】

本発明の実施の形態１におけるソフトウエア開発環境のブロック図

【図２】

同ターゲットプロセッサのブロック図

【図３】

（ａ）同ターゲットプロセッサの命令セットの説明図

（ｂ）同ターゲットプロセッサのレジスタ指定の説明図

【図４】

（ａ）同演算器のモデル図

（ｂ）同メモリコントローラのモデル図

【図５】

同ハードウエアモデルライブラリのヘッダファイルの例示図

【図６】

同ハードウエアモデルライブラリの実装ファイルの例示図

【図７】

同命令セットライブラリの実装ファイルの例示図

【図８】

（ａ）同命令セットライブラリのヘッダファイルの例示図

（ｂ）同ソースコードの例示図

【図９】

（ａ）同インタプリタ型のシミュレータの処理の流れ図

（ｂ）アセンブラによる処理の流れ図

【図１０】

同インタプリタ型のシミュレータの実装ファイルの例示図

【図１１】

本発明の実施の形態２におけるハードウエアモデルライブラリの実装ファイルの例示図

【図１２】

同命令セットライブラリの実装ファイルの例示図

【図１３】

同ソースコードの例示図

【図１４】

本発明の実施の形態３における命令セットライブラリの実装ファイルの例示図

【図１５】

（ａ）同配列の説明図

（ｂ）同ソースコードの例示図

【符号の説明】

１００　リンカ

１０１　ハードウエアモデルライブラリ

１０２　コンパイラ型のシミュレータ

１０３　ソースコード

１０４　コンパイラ

１０５　命令セットライブラリ

１０６　オブジェクトファイル

１０７　オブジェクトコード

１０８　インタプリタ型のシミュレータ

１０９　命令フェッチ手段

１１０　命令デコード手段

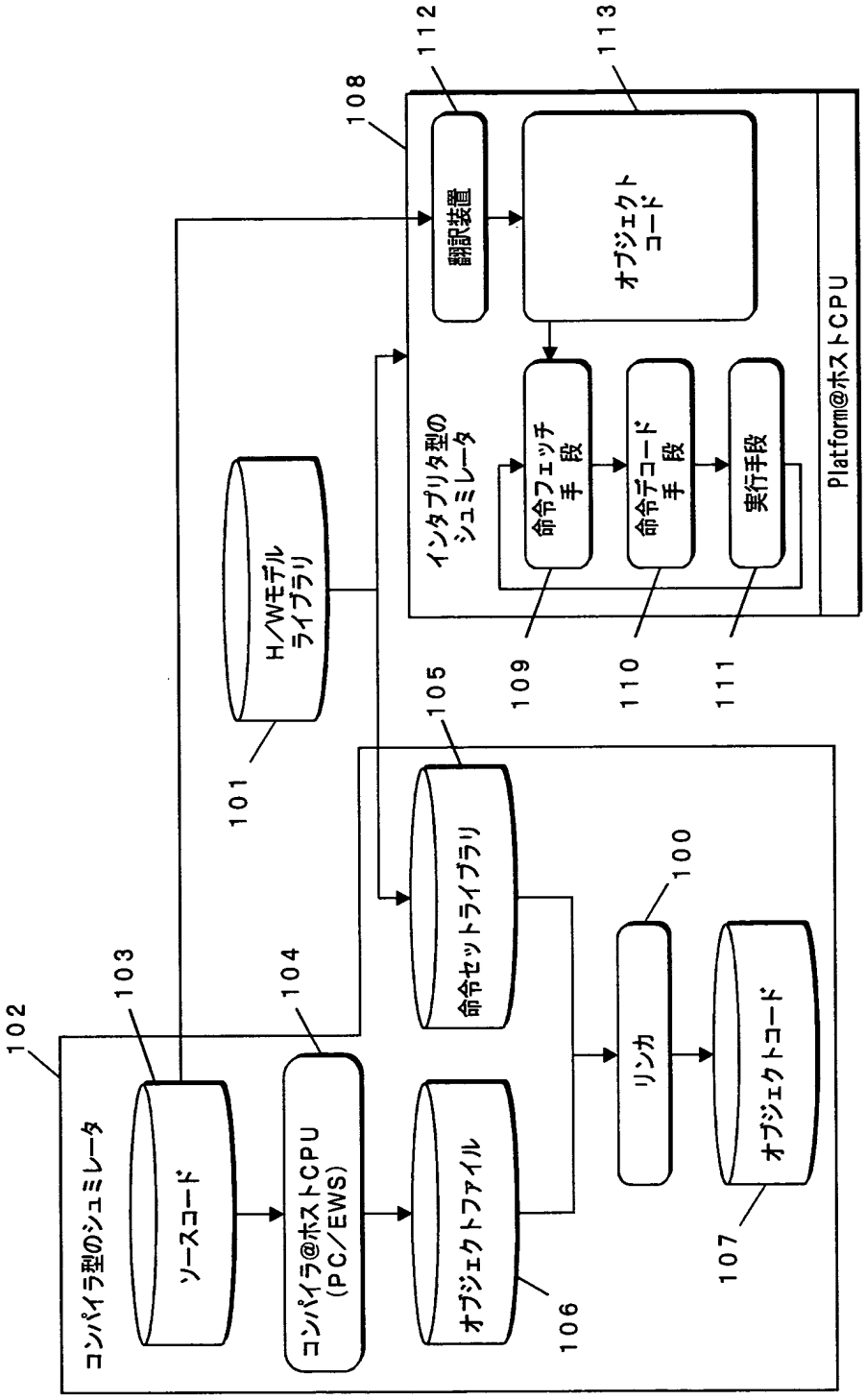
１１１　実行手段

１１２　翻訳装置

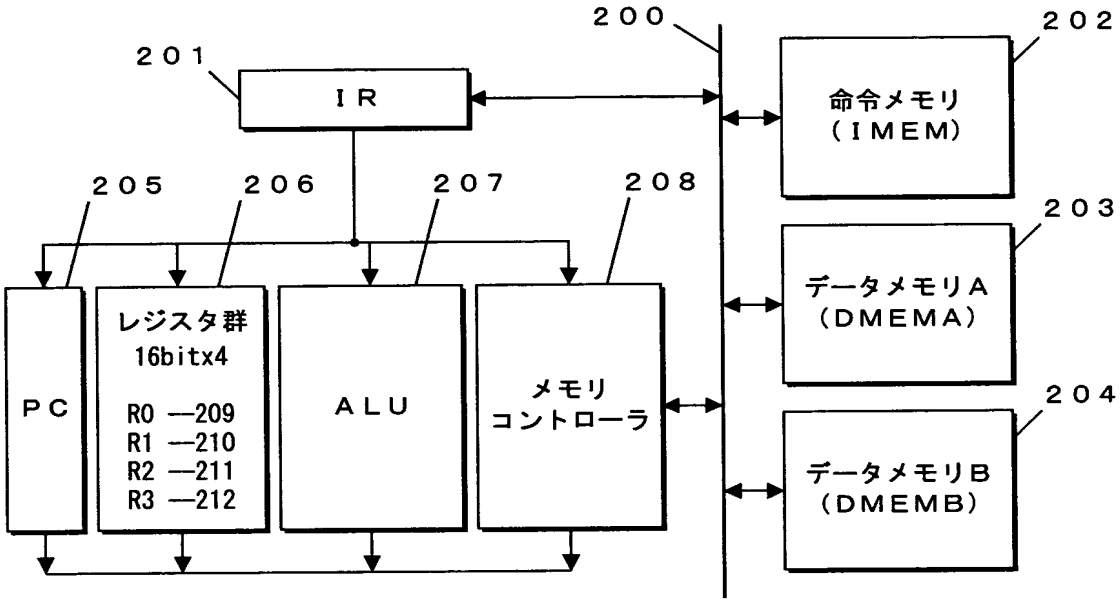
１１３　オブジェクトコード

【書類名】　図面

【図１】


コンパイラ型のシュミレータ
102
ソースコード
103
コンパイラ@ホストCPU
(PC／EWS)
104
オブジェクトファイル
106
命令セットライブラリ
105
H／Wモデル
ライブラリ
101
リンカ
100
オブジェクトコード
107
インタプリタ型の
シュミレータ
108
翻訳装置
112
オブジェクト
コード
113
命令フェッチ
手段
109
命令デコード
手段
110
実行手段
111
Platform@ホストCPU

【図２】



【図３】

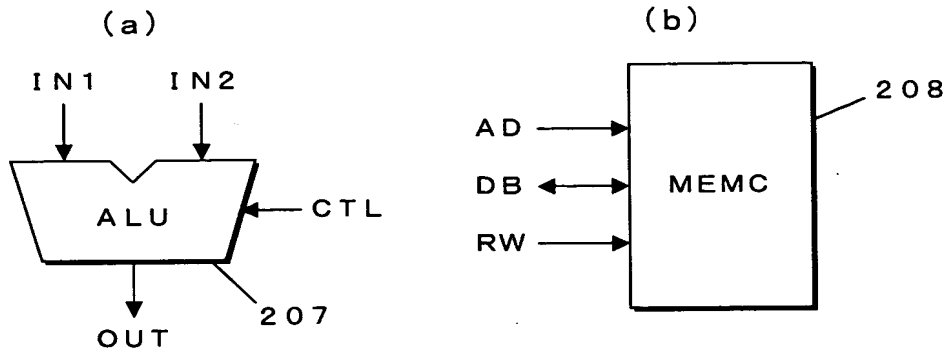
（ａ）

命令	第1 ｵﾍﾟﾗﾝﾄﾞ	第2 ｵﾍﾟﾗﾝﾄﾞ	機械語
ADD	Ra,	Rb	b000
SUB	Ra,	Rb	b001
AND	Ra,	Rb	b010
OR	Ra,	Rb	b011
LD	Ra,	@Rb	b100
ST	Ra,	@Rb	b101
SET	Ra,	IMD	b110
MOV	Ra,	Rb	b111

（ｂ）

レジスタ	機械語
R0	b00
R1	b01
R2	b10
R3	b11

【図４】



【図５】

```
/* hwmodel.h */

extern short  IMEM[N1];
extern short  DMEMA[N2];
extern short  DMEMA[N3];


extern short  IR;
extern short  PC;
extern short  R0;
extern short  R1;
extern short  R2;
extern short  R3;

extern void ALU(short, short, short*, int);
extern void MEMC(short, short*, int);

enum {
  ADD=0, SUB, AND, OR
};
```

【図６】

```
short  IMEM[N1];
short  DMEMA[N2];
short  DMEMB[N3];


short  IR, PC, R0, R1, R2, R3;


void ALU(short IN1, short IN2, short *OUT, int CTL) {
 switch (CTL) {
  case 0;     // ADD
    *OUT = IN1 + IN2;
    break;
  case 1;     // SUB
    *OUT = IN1 - IN2;
    break;
  case 2;     // AND
    *OUT = IN1 & IN2;
    break;
  case 3;     // OR
    *OUT = IN1 ¦ IN2;
    break;
  }
}


void MEMC(short AD, short *DB, int RW){
 switch (RW) {
   case 1: // Read
    if (AD < 0x100)
      *DB = DMEMA[AD & 0xFF];
    else
      *DB = DMEMB[AD & 0xFF];
    break;

   case 0: // Write
    if (AD < 0x100)
      DMEMA[AD & 0xFF] = *DB;
    else
      DMEMB[AD & 0xFF] = *DB;
    break;
  }
}
```

【図７】

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, ADD);
}

void SUB(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, SUB);
}

void AND(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, AND);
}

void OR(short *RS1, short RS2) {
  ALU(*RS1, RS2, RS1, OR);
}

void LD(short *RS1, short RS2) {
  MEMC(RS2, RS1, 1);
}

void ST(short *RS1, short RS2) {
  MEMC(RS2, RS1, 0);
}

void SET(short *RS1, short IMD) {
  *RS1 = IMD;
}

void MOV(short *RS1, short RS2) {
  *RS1 = RS2;
}
```

【図８】

（ａ）

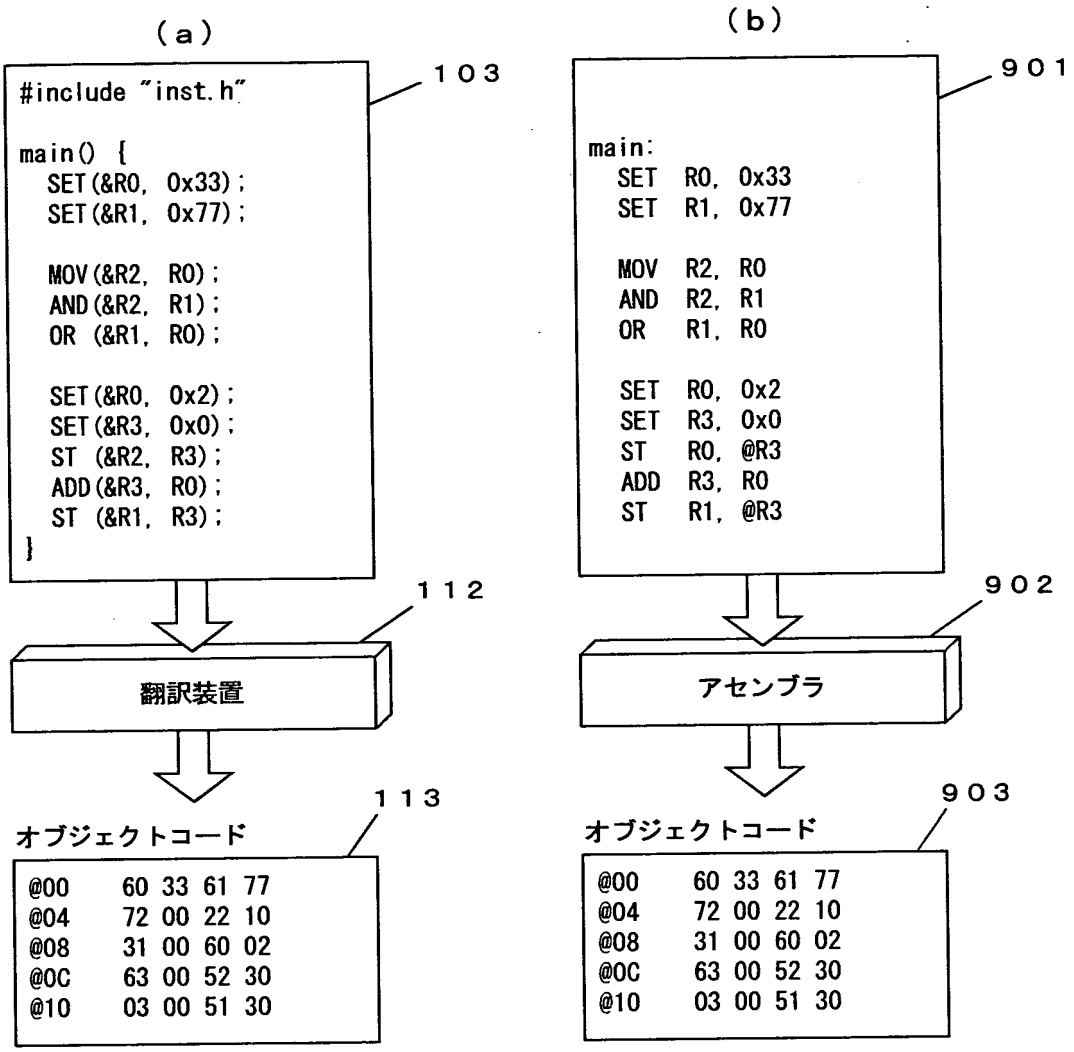
```
/* inst.h */

extern void  ADD(short *, short);
extern void  SUB(short *, short);
extern void  AND(short *, short);
extern void  OR(short *, short);
extern void  LD(short *, short);
extern void  ST(short *, short);
extern void  SET(short *, short);
extern void  MOV(short *, short);
```

（ｂ）

```
#include "inst.h"

main( ) {
      .....

  SET(&R0, 0x33);
  SET(&R1, 0x77);

  MOV(&R2, R0);
  AND(&R2, R1);
  OR (&R1, R0);

  SET(&R0, 0x2);
  SET(&R3, 0x0);
  ST (&R2, R3);
  ADD(&R3, R0);
  ST (&R1, R3);
      .....

}
```

１０３

【図９】


（ａ）
#include "inst.h"
main() {
SET(&R0, 0x33);
SET(&R1, 0x77);
MOV(&R2, R0);
AND(&R2, R1);
OR (&R1, R0);
SET(&R0, 0x2);
SET(&R3, 0x0);
ST (&R2, R3);
ADD(&R3, R0);
ST (&R1, R3);
}
103
112
翻訳装置
113
オブジェクトコード
@00 60 33 61 77
@04 72 00 22 10
@08 31 00 60 02
@0C 63 00 52 30
@10 03 00 51 30
（ｂ）
main:
SET R0, 0x33
SET R1, 0x77
MOV R2, R0
AND R2, R1
OR R1, R0
SET R0, 0x2
SET R3, 0x0
ST R0, @R3
ADD R3, R0
ST R1, @R3
901
902
アセンブラ
903
オブジェクトコード
@00 60 33 61 77
@04 72 00 22 10
@08 31 00 60 02
@0C 63 00 52 30
@10 03 00 51 30

【図１０】

```
#include "hwmodel.h"

enum {
  Fetch=0, Decode, Exec
};

int    cycle;
int    state;
int    *reg[4] = {&R0, &R1, &R2, &R3};

main() {
  cycle = 0;
  state = Fetch;
  while (exec());
  printf("cycle=%d n", cycle);
}

int exec() {
 int cd[3];

 cycle++;

 switch (state) {
  case Fetch:
    IR = IMEM[PC];
    if (IR < 0)
      return 0;
    else {
      PC++;
      state = Decode;
      return 1;
    }
  case Decode:
    cd[0] = (IR>>4)&0x7;
    cd[1] = (IR>>2)&0x3;
    cd[2] = IR & 0x3;
    return 1;
  case Exec:
    if (cd[0]==0x0)
      ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],ADD);
    else if((IR>>4) == 0x1)
      ALU(*reg[cd[1]],*reg[cd[2]],reg[cd[1]],SUB);
          .....
    state = Fetch;
    return 1;
 }
}
```

【図１１】

```
short IMEM[N1];
short DMEMA[N2];
short DMEMB[N3];

short IR, PC, R0, R1, R2, R3;
long cycle;
double power;


void ALU(short IN1, short IN2, short *OUT, int CTL){
  switch (CTL){
  case ADD:
    *OUT = IN1 + IN2;
    break;
  case SUB:
    *OUT = IN1 - IN2;
    break;
  case AND:
    *OUT = IN1 & IN2;
    break;
  case OR:
    *OUT = IN1 | IN2;
    break;
  }
  cycle += 1;
  power += 0.01;
}


void MEMC(short AD, short *DB, int RW){
  switch (RW){
   case 1: // Read
     if (AD < 0x100)
       *DB = DMEMA[AD & 0xFF];
     else
       *DB = DMEMB[AD & 0xFF];
   cycle += 1;
   power += 0.02;
   break;

   case 0: // Write
     if (AD < 0x100)
       DMEMA[AD & 0xFF] = *DB;
     else
       DMEMB[AD & 0xFF] = *DB;
   cycle += 2;
   power += 0.04;
   break;
  }
}
```

【図１２】

```
#include "hwmodel.h"

void ADD(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,ADD);
}

void SUB(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,SUB);
}

void AND(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,AND);
}

void OR(short *RS1, short RS2) {
  ALU(*RS1,RS2,RS1,OR);
}

void LD(short *RS1, short RS2) {
  MEMC(RS2,RS1,1);
}

void ST(short *RS1, short RS2) {
  MEMC(RS2,RS1,0);
}

void SET(short *RS1, short IMD) {
  *RS1 = IMD;
  cycle += 1;
  power += 0.005;
}

void MOV(short *RS1, short RS2) {
  *RS1 = RS2;
  cycle += 1;
  power += 0.005;
}
```

【図１３】

```
#include "inst.h"

main( ) {
      .....
  SET(&R0, 0x33);
  SET(&R1, 0x77);

  MOV(&R2, R0);
  AND(&R2, R1);
  OR (&R1, R0);

  SET(&R0, 0x2);
  SET(&R3, 0x0);
  ST (&R2, R3);
  ADD(&R3, R0);
  ST (&R1, R3);
      .....

  printf(cycle="%d\n",cycle);
  printf(power="%f\n",power);
}
```

【図１４】

```
#include "hwmodel.h"

long   cycle;
long   code;
double power;

void ADD(short *RS1, short RS2){
  ALU(*RS1, RS2, RS1, ADD);
  cycle += cycle_tbl[0];
  power += power_tbl[0];
  code += 2;
}

void SUB(short *RS1, short RS2){
  ALU(*RS1, RS2, RS1, SUB);
  cycle += cycle_tbl[1];
  power += power_tbl[1];
  code += 2;
}

    .....

void LD(short *RS1, short RS2){
  MEMC(RS2, RS1, 1);
  cycle += cycle_tbl[4];
  power += power_tbl[4];
  code += 2;
}

void ST(short *RS1, short RS2){
  MEMC(RS2, RS1, 0);
  cycle += cycle_tbl[5];
  power += power_tbl[5];
  code += 2;
}

    .....

void MOV(short *RS1, short RS2){
  *RS1 = RS2;
  cycle += cycle_tbl[7];
  power += power_tbl[7];
  code += 2;
}
```

【図１５】

（ａ）

命令	index	cycle_tbl[ ]	power_tbl[ ]
ADD	0	1	0.01
SUB	1	1	0.01
AND	2	1	0.01
OR	3	1	0.01
LD	4	2	0.02
ST	5	3	0.03
SET	6	1	0.005
MOV	7	1	0.005

（ｂ）

```
long cycle_tbl[8];
long power_tbl[8];

init(){
      .....
  fp = fopen("table",r);

  for (i=0; i<8; i++)
   fscanf(fp, "%d, %f", &cycle_tbl[i],&power_tbl[i]);
      .....
}
```

【書類名】　要約書

【要約】

【課題】　全て高級言語で開発された１つのソフトウェアを、コンパイラ型、インタプリタ型、シミュレータで、共用できるようにする。

【解決手段】　Ｃ言語のソースコードをコンパイルするコンパイラ１０４と、ターゲットプロセッサのハードウエア構成要素を、Ｃ言語でモデル化する関数等と、この関数等を利用して、ターゲットプロセッサの命令に対応してＣ言語で定義された関数等とを含むライブラリ１０１、１０５を備え、ソースコード１０３は、ライブラリを使用して記述する。

【選択図】　図１

出証特２００３－３０３０３６３

出　願　人　履　歴　情　報

識別番号　　〔０００００５８２１〕

１．変更年月日　　１９９０年　８月２８日
［変更理由］　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社